A Clique Merging Algorithm to Solve Semidefinite Relaxations of Optimal Power Flow Problems

A Clique Merging Algorithm to Solve Semidefinite Relaxations of Optimal Power Flow Problems

Julie Sliwak, Erling D. Andersen, Miguel F. Anjos, *Senior Member, IEEE,* Lucas Létocart, and Emiliano Traversi

***Abstract*—Semidefinite Programming (SDP) is a powerful technique to compute tight lower bounds for Optimal Power Flow (OPF) problems. Even using clique decomposition techniques, semidefinite relaxations are still computationally demanding. However, there are many different clique decompositions for the same SDP problem and they are not equivalent in terms of computation time. In this paper, we propose a new strategy to compute efficient clique decompositions with a clique merging heuristic. This heuristic is based on two different estimates of the computational burden of an SDP problem: the size of the problem and an estimation of a per-iteration cost for a state-of-the-art interior-point algorithm. We compare our strategy with other algorithms on MATPOWER instances and we show a significant decrease in solver time.**

***Index Terms*—Clique Decomposition, Optimal Power Flow, Semidefinite Programming.**

I. Introduction

THE Alternating Current OPF (ACOPF) is a famous power system problem that is highly nonconvex. Although there is still no efficient method to solve this problem to global optimality, all promising methods depend on large-scale SDP problems being solved efficiently.

Sparse large-scale SDPs can be tackled using clique decomposition techniques [1]. Eltved et al. [2] have recently shown the power of these techniques for OPF problems when coupled with an appropriate reformulation. Another enhancement direction consists in computing an appropriate clique decomposition, e.g. using clique merging as Molzahn et al. first proposed for OPF problems in [3]. A few clique merging algorithms have been proposed in the literature for general SDP problems [4], [5], [6]. They all apply the same key idea: traversing the clique tree and merging cliques that share lots of nodes. Molzahn et al. proposed a greedy approach for OPF problems [3] to minimize the size of the SDP problem. This approach turns out to be faster for OPF problems compared to methods like [4]. Our main contribution is to propose a more refined clique merging strategy. It takes into account an estimate of an IP solver per-iteration time, which allows to improve total solution time by 12% for MATPOWER [7] instances.

This paper is organized as follows. Section II describes ACOPF. The standard rank relaxation is presented in section III along with clique decomposition techniques. Section IV details our clique merging strategy. Computational tests are presented in section V and section VI concludes the paper.

J. Sliwak is with RTE, LIPN, UMR CNRS 7030, Université Sorbonne Paris Nord, France and Polytechnique Montréal, Canada.

E. D. Andersen is with MOSEK.

M. F. Anjos is with the University of Edinburgh, United Kingdom.

L. Létocart and E. Traversi are with LIPN, UMR CNRS 7030, Université Sorbonne Paris Nord, France.

II. ACOPF formulation

Notations:

- $T(N, B)$ is a n-buses transmission network;
- $G \subset N$ represents the subset of generator buses;
- At bus $n \in N$, v_n is the complex voltage, S_n the complex power generation, $\mathbf{S_n^l}$ the complex load, $B^-(n)$ the set of entering branches and $B^+(n)$ the set of exiting branches. $\mathbf{v_n^{min}}$, $\mathbf{v_n^{max}}$ are bounds on the voltage magnitude;
- $Re(z)$ (resp. $Im(z)$) denotes the real (resp. imaginary) part of the complex number z;
- For a generator $g \in G$, $\mathbf{c_g}$ and $\mathbf{k_g}$ represent the linear and the constant cost and $\mathbf{P_g^{min}}$, $\mathbf{P_g^{max}}$ (resp. $\mathbf{Q_g^{min}}$, $\mathbf{Q_g^{max}}$) the bounds on the active (resp. reactive) power;
- For a branch b, $i_b^o(v)$ and $i_b^d(v)$ (resp. $S_b^o(v)$ and $S_b^d(v)$) represent the currents (resp. the powers) at the origin and at the destination of the branch. They are linear (resp. quadratic) functions of $v_{o(d)}$ and $v_{d(b)}$ with $o(b)$ and $d(b)$ the ends of branch b. Both currents and powers depend on the admittance matrix $\mathbf{Y_b}$. $\mathbf{i_b^{max}}$ stands for the current magnitude limit for the branch.

The ACOPF problem is defined as follows:

$$
\begin{aligned}
\min_{v,S} \ & \sum_{g \in G} \mathbf{c_g} Re(S_g) + \mathbf{k_g} \\
\text{s.t. } & S_n = \mathbf{S_n^l} + \sum_{b \in B^-(n)} S_b^d(v) + \sum_{b \in B^+(n)} S_b^o(v) && \forall n \in N \\
& (\mathbf{v_n^{min}})^2 \leq |v_n|^2 \leq (\mathbf{v_n^{max}})^2 && \forall n \in N \\
& \mathbf{P_g^{min}} \leq Re(S_g) \leq \mathbf{P_g^{max}} && \forall g \in G \\
& \mathbf{Q_g^{min}} \leq Im(S_g) \leq \mathbf{Q_g^{max}} && \forall g \in G \\
& |i_b^o(v)|^2 \leq (\mathbf{i_b^{max}})^2 && \forall b \in B \\
& |i_b^d(v)|^2 \leq (\mathbf{i_b^{max}})^2 && \forall b \in B \\
& S_n = 0 && \forall n \in N \backslash G \\
& v_n \in \mathbb{C}, \ S_n \in \mathbb{C} && \forall n \in N
\end{aligned}
\tag{1}
$$

where all constant parameters are in bold.

We model thermal limits using current and linear generation costs to reflect the practice at RTE. Moreover, we aggregate generators connected to the same bus using the linear cost of the last generator associated to the bus in the MATPOWER list and the sums of power limits as power bounds. Finally, the power generation variables can be eliminated. The costs

being linear, the resulting problem is a nonconvex Quadratically Constrained Quadratic Problem (QCQP) with complex variables.

III. Rank relaxation and standard resolution

The rank relaxation is the standard SDP relaxation for a nonconvex QCQP as described below. It is obtained by dropping the rank constraint after introducing the complex matrix $W = vv^H$, equivalent to $W \succeq 0$ and $rank(W) = 1$:

$$\left\{ \begin{array}{ll} \min & v^H Q_0 v \\ \mathbf{s.t.} & v^H Q_p v \leq a_p \ \forall p \\ & v \in \mathbb{C}^n \end{array} \right. \Rightarrow \left\{ \begin{array}{ll} \min & Q_0 \cdot W \\ \mathbf{s.t.} & Q_p \cdot W \leq a_p \ \forall p \\ & W \succeq 0 \end{array} \right. \tag{2}$$

This SDP problem has to be converted into a SDP problem with real numbers to be solved. It can be done using the rectangular representation and a matrix of size $2n \times 2n$. To improve resolution, a SDP problem can be reformulated using clique decomposition technique introduced by Fukuda et al. [1]. It results in a problem with several Positive Semidefinite (PSD) constraints on small submatrices plus the required linking constraints between these submatrices. The only requirement to apply clique decomposition is the chordality of the aggregate sparsity pattern G^A, that is the graph representing the nonzeros entries in the SDP data [8]. A graph is chordal if every cycle of length 4 or more has a chord, i.e., an edge joining nonconsecutive vertices. If G^A is not chordal, a chordal extension H has to be computed, usually with a minimal number of additional edges. As finding such a minimal chordal extension is NP-complete [9], several efficient heuristics are available in the literature [10]. The maximal cliques in H, denoted by $L = \{C_1, ..., C_r\}$, define the submatrices that must be PSD in the reformulated SDP problem. A clique is a subset of vertices such that every two distinct vertices in the clique are connected. A clique is maximal if it is not a subset of another clique. Linking constraints between the cliques are specified by a clique tree, i.e., a maximum-weight spanning tree for the clique graph W. The nodes of W are the cliques in L and there is a weighted edge between each pair of vertices that share nodes, its weight being the number of shared nodes.

However, as shown in [11], the computation time is highly sensitive to the choice of the chordal extension heuristic and minimizing the number of additional edges turns out to be inefficient for OPF problems. In particular, [11] presents a clique merging heuristic to confirm that adding more edges allows to improve computation time. Nevertheless, this heuristic is not competitive with the state-of-the-art clique merging heuristic for OPF problems unlike the one presented below.

IV. A new clique merging algorithm

In this section, we present a clique merging algorithm greedily minimizing an IP solver per-iteration time estimation then an estimation of the SDP size.

A. An estimation of an IP solver per-iteration time

For a given decomposition, let us denote $L = \{C_1, ..., C_r\}$ the set of maximal cliques, m the number of constraints in the original SDP problem and ℓ the number of linking constraints coming from the clique tree T. The cardinality of the clique C_i is denoted by $|C_i|$. An estimation of a state-of-the-art IP solver per-iteration time t is given by the formula: $t = \alpha \sum_{i=1}^{r} |C_i|^3 + \beta(m+\ell)^3 + c$ with α, β and c three unknown parameters. This formula is based on a balance between the two algorithmic steps expected to be most computationally expensive. The first term represents the complexity of computing eigenvalues for each submatrix associated to a clique. The second term represents the complexity of the Cholesky factorization for the normal equations.

We determine α, β and c for a given SDP problem by collecting the average time per iteration for several clique decompositions and applying a multilinear regression. We use 10 decompositions with important differences in the number of cliques and of linking constraints to get more diversity. Nine of these decompositions are computed on the SDP relaxation with complex variables, resulting in cliques with complex variables. We double each clique to get cliques with real variables to solve the SDP relaxation with real variables. The tenth one is computed directly on the SDP problem with real variables, using a Cholesky decomposition with an AMD ordering [12]. Among the ones computed on the complex SDP problem, the first two differ in the choice of the chordal extension algorithm: a Cholesky factorization with an AMD ordering for the first one and the Minimum Degree [13] algorithm for the second one. The others are obtained by clique merging algorithms applied on the first decomposition: either the one in [3] or the one in [11] applied once or more (until six times). All SDP problems were solved using MOSEK [14] 9.1.

B. The algorithm

The algorithm proposed in [3] seeks to greedily minimize an estimate of the computational burden of an SDP problem $f_M(L, \ell)$:

$$f_M(L, \ell) = m + \ell + \sum_{C_i \in L} |C_i|(|C_i| + 1)/2 \tag{3}$$

where ℓ depends on the clique tree T:

$$\ell = \sum_{(C_i, C_j) \in E} d_{ij}(2d_{ij} + 1) \tag{4}$$

with d_{ij} the number of complex variables shared by C_i and C_j. Minimizing the function $f_M(L, \ell)$, i.e., the size of the SDP problem, leads to the merging of small cliques. Another reasonable choice consists in minimizing the cost of an IP iteration:

$$f(L, \ell) = \alpha \sum_{C_i \in L} |C_i|^3 + \beta(m + \ell)^3 + c \tag{5}$$

This function is a better estimate of the computational burden but it leads to the merging of large cliques, which is interesting only if we merge a few cliques.

To take advantage of both criteria, we propose to apply the greedy clique merging heuristic described in the pseudo-code in Figure 1: the key idea is to repeatedly merge the two cliques that minimize either our criterion f or the criterion

Input: Cliques in $L = \{C_1, ..., C_r\}$, a clique tree $T = (L, E)$, a parameter k_{max} to switch criterion
Output: A new clique decomposition L, a new clique tree T
$k = 0$
while $|L| \geq 10\%n$ **do**
$k \leftarrow k + 1$
if $k \leq k_{max}$ **then**
Find the edge $(C_i, C_j) \in E$ that minimizes $f(L, \ell)$
else
Find the edge $(C_i, C_j) \in E$ that minimizes $f_M(L, \ell)$
end if
$L \leftarrow (L - \{C_i, C_j\}) \cup \{C_i \cup C_j\}$
Merge the nodes C_i and C_j in T
Update the number of linking constraints ℓ
end while

Fig. 1. Clique merging algorithm for a n-bus instance

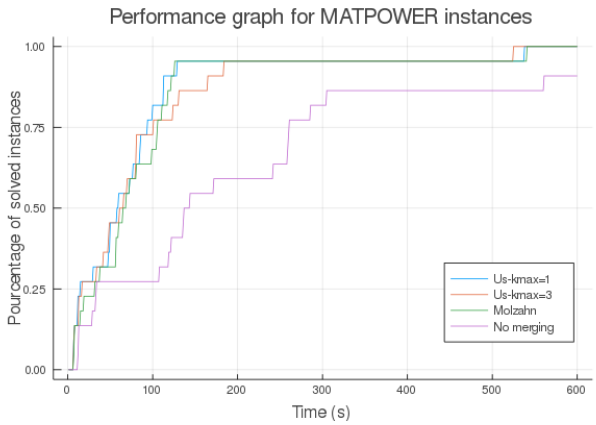

Fig. 2. Performance graph for MATPOWER instances with more than 1000 buses (except ACTIVSG25k) for four strategies: our strategy with $k_{max} = 1$ and $k_{max} = 3$, the strategy in [3], and no clique merging.

f_M as long as the number of cliques is greater than 10% of the number of buses. This value of 10% was recommended in [3]. Adjusting the switching parameter k_{max} should allow us to compute decompositions with less linking constraints and reasonably sized cliques. In practice, we observe a reduction in computation time for small values of k_{max}. Note that we compute both clique decompositions for the complex SDP relaxation. Before the clique merging heuristic, we use a Cholesky factorization with an AMD ordering to compute a chordal extension and Prim's algorithm for the clique tree.

V. Computational results

We tested our strategy with several values of k_{max} on the MATPOWER [7] instances with more than 1000 buses. The tests were carried out on a Processor Intel® Core™ i7-6820HQ CPU @2.70GHz. We used Julia 1.0.3. [15], JuMP.jl [16], LightGraphs.jl [17], and Mosek.jl with MOSEK 9.1 [14].

We compared our strategies to the one presented in [3] and to a basic strategy without clique merging. We chose the heuristic in [3] as a comparator because it is the fastest heuristic that has been proposed for OPF problems. All chordal extensions have been computed with a Cholesky factorization and an AMD ordering. We implemented the strategy in [3] in our Julia setting for a fair comparison. For $k_{max} \geq 5$, there is no gain in computation time compared to [3] but an improvement is observed for k_{max} between 1 and 4. The most significant improvements are observed with $k_{max} = 1$ and $k_{max} = 3$. The results are reported in the performance graph in Figure 2. This graph presents the percentage of solved instances at each second for the four strategies (ours with $k_{max} = 1$ and $k_{max} = 3$, the one in [3] and no clique merging). Most of the instances are solved within a hundred seconds for the three clique merging strategies whereas only the half is solved for the basic strategy. Our strategy with $k_{max} = 1$ solves more instances than the one in [3] at each time step. Our strategy with $k_{max} = 2$ solves more instances than the one in [3] at each time step under 100 seconds. For more details, Table I compares computation time for the clique merging strategies for k_{max} from 1 to 4 on each instance. For 22 out of 23 instances, one of our strategies is faster than the one in [3]. However, there are differences in performance depending on k_{max}. The strategy with $k_{max} = 1$ seems to be the most efficient since it is faster than the one in [3] with a reduction of 12% in the total time for the 23 instances, and it significantly decreases the solution time for the largest instances. It is also the most reliable strategy since 19 out of 23 instances are improved. The other values of k_{max} also improve the total computation time but they mainly improve the largest instance ACTIVSg25k. Indeed, the improvement in the total computing time for instances of less than 20,000 buses is 4% or less. Besides, the strategies for k_{max} between 2 and 4 seem more unstable: they can improve as well as deteriorate computation time. For example, for $k_{max} = 3$, computation time is decreased by 21% for instance 6515rte whereas it is is increased by 36% for instances 6468rte and 6495rte. For that reason, we would recommend to use our strategy with $k_{max} = 1$. Nevertheless, we would recommend to test $k_{max} = 3$ for instances with more than ten thousands buses since the gain in computing time is more interesting in this case.

VI. Conclusion

We presented a strategy to compute good clique decompositions for OPF problems. This strategy is based on a clique merging algorithm that minimizes greedily either an estimate of an IP solver per-iteration time or the SDP size. Using this strategy with a relevant switching parameter k_{max} leads to a reduction in total computation time of 12%. This strategy could be extended for generic sparse SDP problems for which the decomposition contains lots of small cliques. Future work will consider a chordal extension heuristic that provides clique decompositions similar to those obtained by clique merging.

References

[1] M. Fukuda, M. Kojima, K. Murota, and K. Nakata, "Exploiting sparsity in semidefinite programming via matrix completion i: General framework," *SIAM Journal on Optimization*, vol. 11, no. 3, pp. 647–674, 2001.

TABLE I
COMPARISON OF MOSEK SOLUTION TIME

Instance	Time (s)				
	$k_{max}=1$	$k_{max}=2$	$k_{max}=3$	$k_{max}=4$	[3]
1354pegase	**6.88**	8.22	7.28	7.05	7.19
1888rte	6.8	6.73	6.76	**6.51**	6.59
1951rte	7.06	6.91	6.88	**6.63**	7.55
ACTIVSg2000	**76.59**	88.67	80.56	80.88	80.61
2383wp	**29.86**	35.41	33.95	34.31	37.50
2736sp	49.70	48.48	48.47	**47.45**	64.61
2737sop	47.83	46.08	41.39	**40.63**	56.39
2746wop	72.23	62.83	60.64	**59.27**	68.5
2746wp	49.2	**47.78**	47.89	48.56	56.73
2848rte	**11.22**	12.38	12.28	11.52	14.09
2868rte	**14.3**	15.91	16.02	15.64	18.11
2869pegase	**11.59**	12.63	12.63	12.48	31.17
3012wp	59.88	67.58	65.67	**57.53**	72.5
3120sp	84.06	76.00	79.23	**71.55**	110.98
3375wp	**57.28**	72.34	69.05	61.19	59.55
6468rte	**93.13**	109.63	164.50	108.22	105.84
6470rte	112.48	119.97	123.20	119.50	**104.55**
6495rte	**99.92**	119.38	183.06	173.06	117.03
6515rte	113.00	103.70	**100.44**	117.00	121.91
9241pegase	128.59	131.22	130.58	**122.69**	125.89
ACTIVSg10k	537.63	596.31	**524.05**	548.81	540.30
13659pegase	85.31	82.88	80.70	**79.75**	98.67
ACTIVSg25k	10153.39	10724.70	**10029.44**	10441.20	11377.30
Total	**11907.93**	12595.74	11924.67	12271.43	13283.56
Improvement wrt [3]	12%	5%	11%	8%	
Nb improved instances	19/23	14/23	16/23	17/23	
Total without ACTIVSg25k	1754.54	1871.04	1895.23	1830.23	1906.26
Improvement wrt [3]	9%	2%	1%	4%	

[2] A. Eltved, J. Dahl, and M. S. Andersen, "On the robustness and scalability of semidefinite relaxation for optimal power flow problems," *Optimization and Engineering*, pp. 1–18, 2018.

[3] D. K. Molzahn, J. T. Holzer, B. C. Lesieutre, and C. L. DeMarco, "Implementation of a large-scale optimal power flow solver based on semidefinite programming," *IEEE Transactions on Power Systems*, vol. 28, no. 4, pp. 3987–3998, 2013.

[4] K. Nakata, K. Fujisawa, M. Fukuda, M. Kojima, and K. Murota, "Exploiting sparsity in semidefinite programming via matrix completion ii: Implementation and numerical results," *Mathematical Programming*, vol. 95, no. 2, pp. 303–327, 2003.

[5] K. Fujisawa, M. Fukuda, and K. Nakata, "Preprocessing sparse semidefinite programs via matrix completion," *Optimization Methods and Software*, vol. 21, no. 1, pp. 17–39, 2006.

[6] Y. Sun, M. S. Andersen, and L. Vandenberghe, "Decomposition in conic optimization with partially separable structure," *SIAM Journal on Optimization*, vol. 24, no. 2, pp. 873–897, 2014.

[7] R. D. Zimmerman, C. E. Murillo-Sánchez, R. J. Thomas *et al.*, "Matpower: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Transactions on power systems*, vol. 26, no. 1, pp. 12–19, 2011.

[8] L. Vandenberghe, M. S. Andersen *et al.*, "Chordal graphs and semidefinite optimization," *Foundations and Trends in Optimization*, vol. 1, no. 4, pp. 241–433, 2015.

[9] M. Yannakakis, "Computing the minimum fill-in is np-complete," *SIAM Journal on Algebraic Discrete Methods*, vol. 2, no. 1, pp. 77–79, 1981.

[10] P. Heggernes, "Minimal triangulations of graphs: A survey," *Discrete Mathematics*, vol. 306, no. 3, pp. 297–317, 2006.

[11] J. Sliwak, M. Anjos, L. Létocart, J. Maeght, and E. Traversi, "Improving clique decompositions of semidefinite relaxations for optimal power flow problems," EasyChair Preprint no. 2546, EasyChair, 2020.

[12] P. R. Amestoy, T. A. Davis, and I. S. Duff, "An approximate minimum degree ordering algorithm," *SIAM Journal on Matrix Analysis and Applications*, vol. 17, no. 4, pp. 886–905, 1996.

[13] A. Berry, P. Heggernes, and G. Simonet, "The minimum degree heuristic and the minimal triangulation process," in *International Workshop on Graph-Theoretic Concepts in Computer Science*. Springer, 2003, pp. 58–70.

[14] Mosek, "The mosek optimization software," *Online at http://www. mosek. com*, 2019.

[15] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman, "Julia: A fast dynamic language for technical computing," *arXiv preprint arXiv:1209.5145*, 2012.

[16] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM Review*, vol. 59, no. 2, pp. 295–320, 2017.

[17] S. Bromberger, J. Fairbanks, and other contributors, "Juliagraphs/lightgraphs.jl: an optimized graphs package for the julia programming language," 2017. [Online]. Available: https://doi.org/10.5281/zenodo.889971